

Figure 4.9: SA with 196 samples.

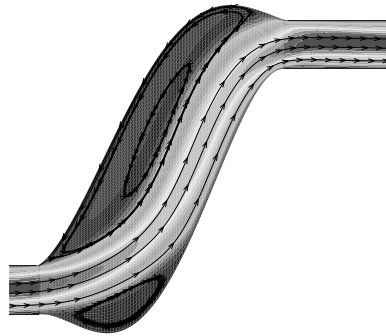


Figure 4.10: Best result by varying parameters. Pressure drop at 5.7698.

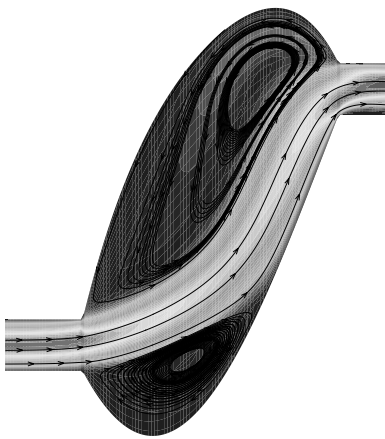


Figure 4.11: 12-Neuron Network fed with 243 samples.

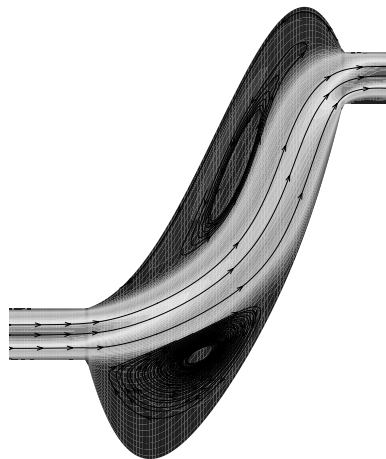


Figure 4.12: 24-Neuron Network fed with 243 samples.

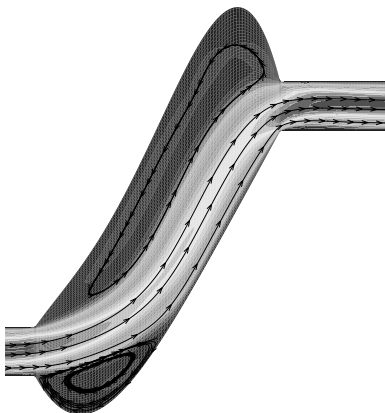


Figure 4.13: 48-Neuron Network fed with 243 samples.

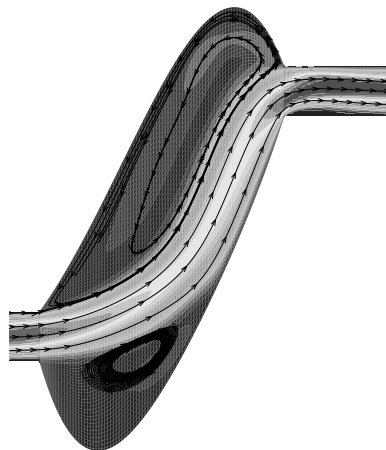


Figure 4.14: 48-Neuron Network fed with 100 samples.

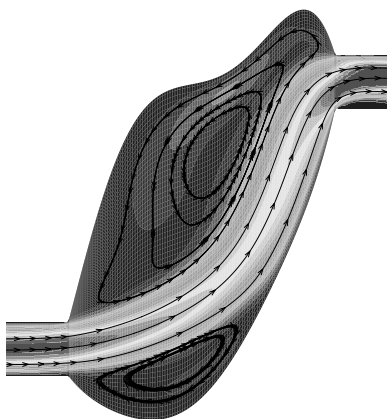


Figure 4.15: 12-Neuron Network fed with 104 samples.

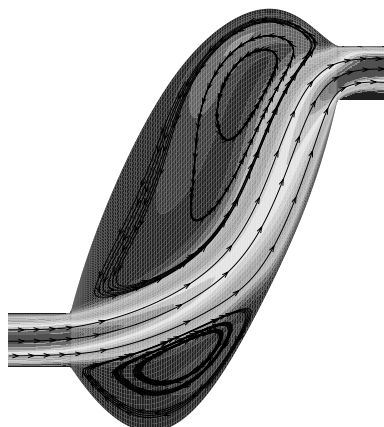


Figure 4.16: 24-Neuron Network fed with 104 samples.

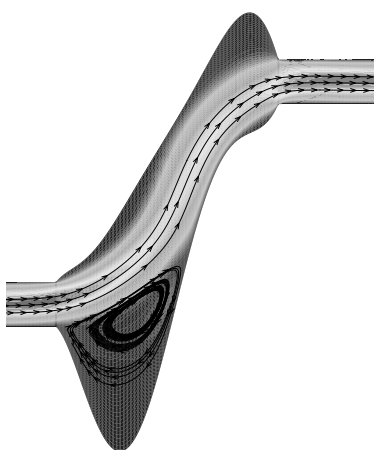


Figure 4.17: SA with 215 samples.

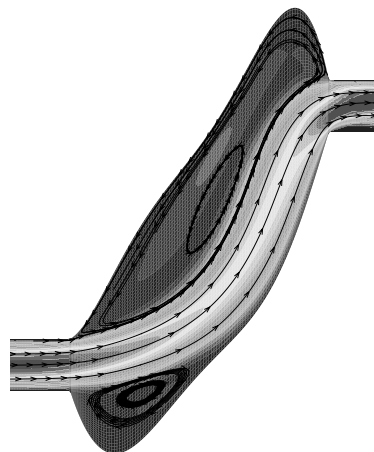


Figure 4.18: 48-Neuron Network fed with 220 samples.

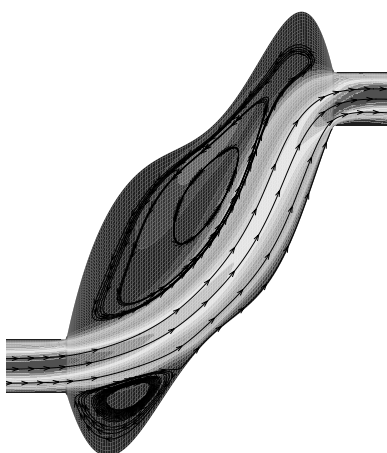


Figure 4.19: 24-Neuron Network fed with 415 samples.

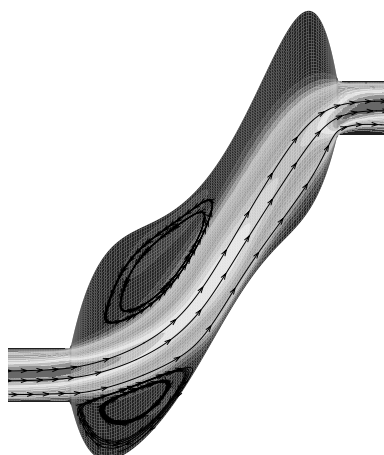


Figure 4.20: 12-Neuron Network with 248 training samples.

account for the low dependence of the corresponding design variable on the objective function, the pressure drop. Lehnhäuser [96] and Hahn [57] also discuss the physics of this specific configuration and argument with side friction effects to explain the pressure drop behavior in dependence on dissipated energy from recirculation zones. The cited literature is a more detailed investigation concerning the physical effects of the presented configuration.

Table 4.6 presents the data sets used for the more complex geometry parameterized by 12 design variables. This configuration is shown in figures (4.15)-(4.14) corresponding to tables 4.7 - 4.10. We here also employed the ENF algorithm which gave surprisingly good results compared to the NF algorithm. As in the 6-parameter example, the SA algorithm outperformed all other algorithms. Again, the two-layer networks only converged for 12/2 neurons. Tables 4.7-4.10 also indicate the network performance for different network and data set configurations.

The choice of generations and individuals (30/30) proved to be appropriate for attaining efficiently the best possible shape design. With respect to find a network as small and simple as possible the network with 48 neurons appeared to be the most promising. 12- and 24-neuron networks also gave reasonable results. We will present our conclusions to this example together with the numerical investigations performed in the next section.

4.3 ANFIS

We introduce adaptive neuro-fuzzy inference systems (ANFIS) which consist of a fuzzy inference system exposed to neural network methodology. These very recent techniques have been developed and elaborated independently by Roger Jang [82, 83] and simultaneously by Lin and Lee [97] and Wang and Mendel [155]. Applications can be found in [12, 62, 89, 153, 157], where we can say that these concept are not yet thoroughly investigated for parameter optimization purposes. The following section will stay close to the methodology from the cited literature.

The present ANFIS approach consists of a fuzzy inference system (FIS) which is a ‘deduction’ machine based on fuzzy logic. Fuzzy logic is a tool based on ideas from fuzzy sets and serve for representing and utilizing data and information that possess non-statistical uncertainty. Fuzzy inference is the process of formulating the mapping from a given input to an output using fuzzy logic [83]. The mapping then provides a basis from which patterns can be discerned. This will later serve for our optimization approach, i.e. this “inference machine” is used as a substitute model to function evaluations.

The FIS is mapped into a complete neural network, giving the adaptive neuro-fuzzy inference system. The ANFIS is a particular neural network and may so provide algorithms for problems in optimization, classification, and clustering of data [19, 89, 90]. The term ‘neuro-fuzzy’ modeling means that various learning techniques developed in the neural network literature are now applicable to fuzzy inference systems: Once a fuzzy inference system is equipped with parameter adaption capabilities, all the design methodologies for neural networks become directly applicable to fuzzy inference systems. As a result, a fuzzy inference system can now adapt itself using numerical data to achieve better performance.

ANFIS found numerous applications in signal processing, time-series prediction, automatic control, data classification, decision analysis, and computer vision [89, 90]. In our context, neural networks are employed to approximate the ‘response surface’ of the target function to be minimized. When an ANFIS model is attained that simulates the target function surface, we can easily apply any optimization tool which uses the model for function evaluations. These evaluations are just linear operations and, therefore, can be performed much more quickly than the usually very time consuming flow computations with, for instance, a finite-volume solver. The ANFIS can be considered as a progressive and ‘high-end’ network model for usage in function approximating. To our best knowledge, the power of this network model has not yet been explored by applications in context of engineering implementations.

We give a short introduction to fuzzy set theory, line out fuzzy inference systems, and then show how to tackle the parameter identification problem in a fuzzy inference system using concepts from neural networks. The proposed methods are validated by numerical examples.

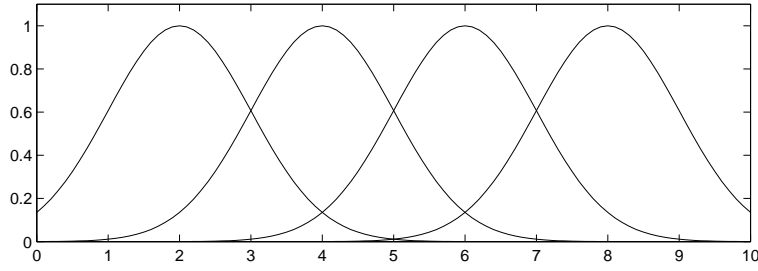


Figure 4.21: Gaussian initial membership functions for an inference system.

4.3.1 Introduction to Fuzzy Sets and Fuzzy Logic

The idea of fuzzy logic is to admit a ‘truth status’ between purely true and false. Naturally, statements in real life are not either perfectly right (true) or wrong (false), but may be seen as true (false) for the most part of it. In 1965, Lofti Zadeh [159, 160] published his pioneering work on what he called fuzzy logic - a part of set theory that operated over the range $[0,1]$ for possible logical states.

From a set theory point of view, a fuzzy set is an extension of a crisp set. Crisp sets allow only for full membership or no membership at all, expressed via the characteristic function

$$\nu_A(x) := \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

In a fuzzy set A , a membership function $\mu_A(x)$ determines the degree of membership of x to a set A . $\mu_A(x)$ takes values in the interval $[0,1]$ and thus logical states between true and false, ‘belonging to a set’ and ‘not belonging to a set’. Fuzzy sets represent linguistic labels like high, low, tall, small, hot, cold, slow, fast etc. So the transition from “belonging to a set” and “not belonging to a set” is gradual, characterized by $\mu_A(x)$. The membership function maps each element of the universe of discourse X to a membership grade between 0 and 1. The value zero indicates complete non-membership, the value one indicates complete membership, and values in between are used to represent intermediate degrees of membership.

There are different kinds of membership functions associated with each input and output response. Some features to note are shape, height, width, and overlap. The most popular membership function is the Gaussian membership function (see figure (4.21))

$$\mu_G(x; \sigma, c) = e^{-\left(\frac{x-c}{\sigma}\right)^2},$$

where c represents the center and σ the width of the Gaussian curve.

In fuzzy logic, truth is a matter of degree. In order to define fuzzy logic operators, we have to find the corresponding operators that preserve the homeomorphism between classical set theory and propositional logic. The important thing in defining fuzzy set logical operators is that if we keep fuzzy values to the extreme 1 (true) or 0 (false) the standard logical operations should hold.

We define operations on fuzzy sets such that the corresponding crisp logical AND, OR, and NOT operators are preserved. First, the fuzzy subset A of a set X , denoted by $A \subset_F X$, is defined to be the set of tuples

$$A := \{(x, \mu_A(x)) | x \in X\}.$$

Equality and inclusion are also defined through the membership function $\mu_A(x)$, so two fuzzy sets $A \subset_F X$ and $B \subset_F X$ are equal if

$$A = B \Leftrightarrow \mu_A(x) = \mu_B(x) \quad \forall x \in X.$$

Accordingly, a fuzzy set A is a subset of a fuzzy set B if

$$A \subset B \Leftrightarrow \mu_A(x) \leq \mu_B(x) \quad \forall x \in X.$$

We can now say that crisp sets are special cases of fuzzy sets since, if the membership function is restricted to the values 0 and 1, the fuzzy set reduces to its crisp counterpart. We also define set operations like union and intersection for fuzzy sets. This is captured by the corresponding set operators AND, OR, and NOT by min, max, and complement as follows

$$\begin{aligned} \mu_{A \cup B}(x) &= \max(\mu_A(x), \mu_B(x)) \\ \mu_{A \cap B}(x) &= \min(\mu_A(x), \mu_B(x)) \\ \mu_{\bar{A}}(x) &= 1 - \mu_A(x). \end{aligned}$$

Most applications use min for fuzzy intersection, max for fuzzy union, and the complement operation for counterpart. These operators (necessarily) reduce to their crisp logic counterparts when the membership functions are restricted to 0 or 1. However, min and max are not the only way to describe the intersection and union of two sets. Zadeh, for instance, defined fuzzy union and fuzzy intersection as the probabilistic AND and OR,

$$\begin{aligned} \mu_{A \cap B}(x) &= \mu_A(x)\mu_B(x), \\ \mu_{A \cup B}(x) &= \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x). \end{aligned}$$

It is noteworthy that the constituent members of the fuzzy set stem from subjective, imprecise or uncertain statements where the membership grade is inherent (determined) by the set rather than random. The above rules give us a tool to work with sets that consist of partial members. The next section explains how these rules find application as a deduction (or: inference) machinery.

4.3.2 Fuzzy if-then Rules and Fuzzy Inference

From Kulkarni [90], Kulkarni and Cavanaugh [89], Jang [82] and Jang and Sun [83], we briefly review the basic concepts of fuzzy inference systems and their adaptive-neuro counterparts. A more detailed presentation is available in the literature.

By the application of fuzzy logical and set operations we can establish fuzzy inference rules. The mapping mechanism is based on these inference rules which are set up as usual inference rules, based on case differentiation. Nonetheless, they differ in semantics: A fuzzy if-then rule assumes the form

$$\text{if } x \text{ is } A \text{ then } y \text{ is } B, \quad (4.5)$$

where A and B are linguistic values defined by fuzzy sets on universes of discourse X and Y , respectively. “ x is A ” is called the antecedent or premise while “ y is B ” is called the consequence or conclusion.

Equation (4.5) is an example for the Mamdani fuzzy inference method, which is the most commonly seen fuzzy methodology. Mamdani-type inference expects the output membership functions to be a fuzzy set: “ y is B ”. If the conclusion is a linear function, the fuzzy model is called a Sugeno fuzzy model. A typical rule in a Sugeno fuzzy model (Takagi and Sugeno [150]) with two inputs x_1 and x_2 has the form

$$\text{If } x_1 \text{ is } A_1 \text{ AND } x_2 \text{ is } A_2, \text{ then } z = px_1 + qx_2 + r$$

The parameters p , q , and r are initially guessed (cf. network initialization) and then adjusted by network training. For a zero-order Sugeno model, the output level z is a constant ($p = q = 0$).

Technically, it is not easy to identify the fuzzy rules and adjust the membership functions. For this it is necessary to formalize the “if-then rules” and to employ a proper algorithm to learn the mapping performance. Here, fuzzy inference systems show a similar structure as neural networks: Both methods tune parameters until they map the desired functional behavior by using ‘feedforward’ and ‘backpropagation’ ideas.

Several types of fuzzy reasoning have been proposed in the literature, depending on the types of fuzzy if then-rules employed. Since we are aiming to employ Sugeno’s fuzzy inference rules, the output of each rule is a linear combination of input variables plus a constant term, and the final output is the weighted average of each rule’s output. The essential rationale behind fuzzy reasoning is a generalized fuzzy inference system which is composed of five functional blocks (cf. figure (4.22)), roughly described by

- A *rulebase* containing a number of fuzzy if-then rules. As the heart of the FIS, the input-output map may be seen as established when the rules have once been set up.
- A *database* which defines the membership functions of the fuzzy sets used in the fuzzy rules.
- A *decision-making* unit performing the inference operations (fuzzy Reasoning). This defines a mapping from input fuzzy sets into output fuzzy sets. It determines the degree to which the antecedent is satisfied for each rule.
- A *fuzzyfication-* and *defuzzyfication interface* which maps input numbers into corresponding fuzzy memberships and fuzzy output into crisp numbers, thus moving from a fuzzy set to a crisp set and vice versa.

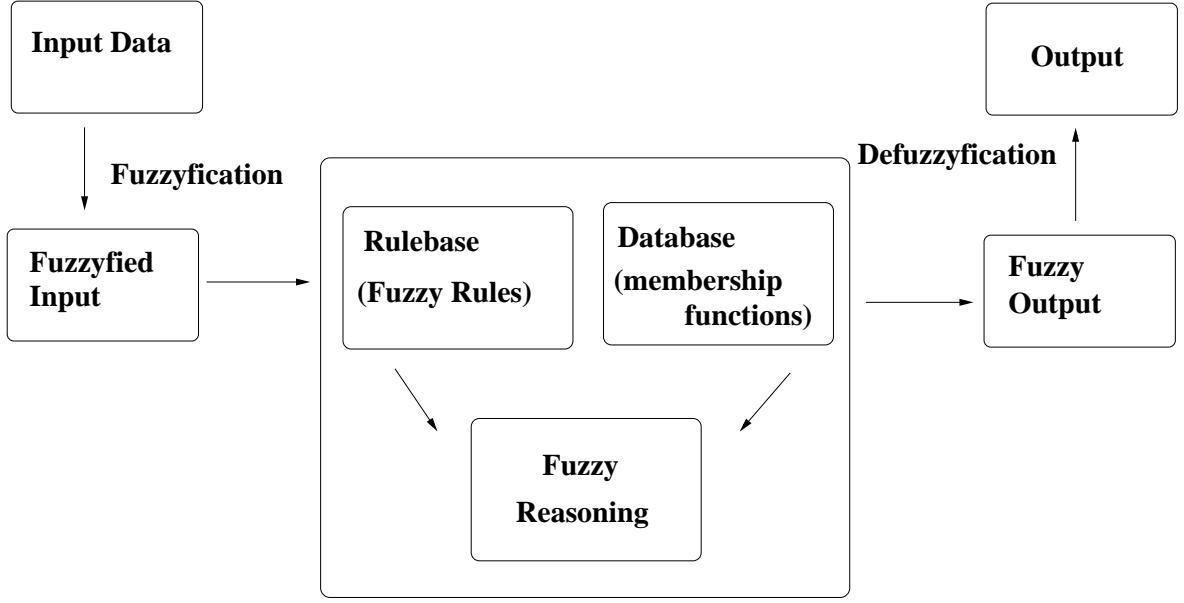


Figure 4.22: A fuzzy inference system

There are five steps to be performed in a fuzzy inference system (cf. figure (4.23)). First, for simplicity we assume the fuzzy inference system under consideration has two inputs x_1 and x_2 . For a first order Sugeno fuzzy model, a typical rule set with two fuzzy if-then rules can be expressed by

Rule 1: If x_1 is A_1 and x_2 is B_1 , then $f_1 = p_1x_1 + q_1x_2 + r_1$,

Rule 2: If x_1 is A_2 and x_2 is B_2 , then $f_2 = p_2x_1 + q_2x_2 + r_2$.

The fuzzy inference model has the following layers, constituting the ANFIS architecture:

Layer 1 The first step is to take inputs and determine the degree to which they belong to each of the fuzzy sets via membership functions (fuzzyfication). The output for units in this layer is given by

$$\begin{aligned} out_i^1 &= \mu_{A_i}(x) \text{ for } i = 1, 2 \\ out_i^1 &= \mu_{B_{i-2}}(x) \text{ for } i = 3, 4 \end{aligned}$$

where x is the input to node i and A_i is a fuzzy set associated with this node. Nodes in this layer represent the linguistic variables.

Layer 2 The Π nodes employ the AND operator and use the product

$$out_i^2 = w_i := \mu_{A_i \cup B_i}(x) = \max_{x \in A_i \cup B_i} (\mu_{A_i}(x), \mu_{B_i}(x)).$$

So the output of each unit in this layer is a product of the incoming signals. The number of fuzzy rules in the inference system is essentially determined by the number of units in this layer. The w_i are also called firing strengths and nodes in this layer are called linguistic operators.

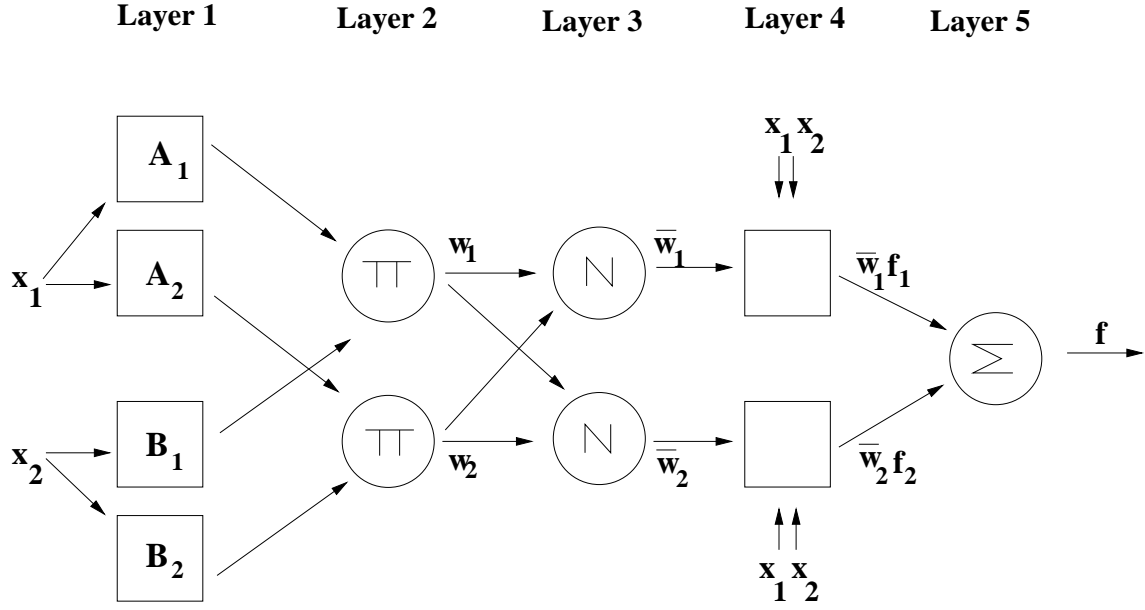


Figure 4.23: A Sugeno type fuzzy inference system with two rules for each input.

Layer 3 Units in this layer calculate the firing strengths of all rules by

$$out_i^3 = \bar{w}_i = \frac{w_i}{\sum_{j=1}^R w_j},$$

where R is the number of rules in the system. Outputs of this layer will be called normalized firing strengths.

Layer 4 Here we generate the consequence of each rule depending on the firing strength. Each node calculates the output on the basis of the first-order polynomial equation. For our Sugeno model, the output for unit i is given by the node function

$$out_i^4 = \bar{w}_i f_i = \bar{w}_i (p_i x_1 + q_i x_2 + r_i)$$

where \bar{w}_i is the output of layer 3 and $\{p_i, q_i, r_i\}$ the parameter set.

Layer 5 This unit aggregates the qualified consequences to produce a crisp output (defuzzification). The remarkable feature of Sugeno fuzzy systems is that defuzzification is computationally cheap: The output for the unit in this layer is given by

$$out^5 = \text{overall output} = f = \sum_{i=1}^R \bar{w}_i f_i = \frac{\sum_{i=1}^R w_i f_i}{\sum_{j=1}^R w_j}$$

where \bar{w}_i represents the normalized firing strength. The indicated defuzzification method is equivalent to the more general center of area defuzzification method which is most frequently used but computationally demanding for non-Sugeno type systems.

4.3.2.1 Adaptive Neuro-Fuzzy Inference System

The acronym ANFIS derives its name from adaptive neuro-fuzzy inference system. The basic idea behind these neuro-adaptive learning techniques is very simple. Using a given input/output data set, the ANFIS algorithm constructs a fuzzy inference system whose membership function parameters are adjusted using a backpropagation algorithm. This learning method is very similar to that of neural networks. In fact, the computing frameworks of radial basis function networks and ANFIS for the zero-order Sugeno model are functionally equivalent [84].

The parameters associated with the membership functions will change through the learning process. The computation of these parameters (their adjustment) is facilitated by a gradient vector, usually the derivative of the performance (or error) function which computes the summed squared distance of the network output to the target training values. Once the gradient vector is obtained, any of several optimization routines could be applied in order to adjust the parameters to reduce an error measure.

To recap, a fuzzy system consists of several rules of which each rule has a premise and a consequence part. The premise contains the initial parameters and defines a conjunction of certain fuzzy operators over the rule space in the input-output space. The consequence consists of a linear equation for the Sugeno-type model.

The parameter identification problem can now be described as follows: First, a determination of the number of rules and initial variables in the premise (structure identification) has to be made. This will be captured by a clustering algorithm (Chi [19]) for which we point to the next subsection. Second, we consider the parameter optimization problem of the premise and consequence parameters. Both tasks can be coped separately. By fixing the structure and the premise parameters, and since we have for the system output

$$f = \frac{\sum_{i=1}^R w_i f_i}{\sum_{j=1}^R w_j} = \sum_{i=1}^R \bar{w}_i f_i = \sum_{i=1}^R \bar{w}_i (p_i x + q_i y + r_i), \quad (4.6)$$

where f now became the objective of the optimization procedure to minimize the difference between training samples and network output. In (4.6) we see that finding the consequence parameters (p_i , q_i , and r_i) is a linear optimization problem, to be accomplished by a least-squares approach. During the second phase of learning (the backward pass), the consequence parameters are kept fixed and the premise parameters are solutions of a nonlinear optimization problem which can be approached by a gradient descent method.

The modeling approach used by ANFIS is similar to many system identification techniques (e.g. neural networks): First, one sets up a parametrized model structure. The next step is to collect input/output data in a form that will be usable by ANFIS for training. Then a network training methodology is used to train the FIS model in order to emulate the training data presented to it by modifying the membership function parameters according to a chosen error criterion. The network-type structure can then be used to interpret an input/output map [89].

In general, this type of modeling works well, if the training data presented to the ANFIS

for training (estimating) membership function parameters is fully representative of the features of the data that the trained FIS is intended to model. However, this is not always the case. The input/output samples may be numerically perturbed, e.g. by noise. Our purpose is to use ANFIS by ‘training’ it with the least necessary number of samples. This may erroneously bias the mapping abilities of the inference machine. We thus need to resort to validation techniques which ensure a proper data classification of the fuzzy inference machine.

4.3.2.2 Model Validation and Clustering

Model validation is the process by which the input vectors from input/output data sets on which the FIS was **not** trained, are presented to the trained FIS model to see how well it predicts the corresponding data set output values.

This type of validation data set is used to control the potential for overfitting the data. When validation data is presented to ANFIS as well as training data, the FIS model selects the parameters associated with the minimum validation data error. Here, the error is always defined to be the sum squared error between predicted and actual system response. The initial FIS structure is based on a fixed number of membership functions. Depending on the complexity of the underlying problem, this may invoke the so-called curse of dimensionality which causes an explosion of the number of rules when the number of inputs is moderately large, that is, more than four or five. To antagonize this effect, a subtractive clustering algorithm is employed.

Clustering is used to identify groupings of data in the premise variable space. For a large data set, this serves to find a concise representation of a system’s behavior. In a network system, this technique is used to model the input/output coherence having a minimum network complexity. Each data point in the output space belongs to a set of nearby data points and forms a cluster, depending on a given clustering radius (this technique is detailed in Bedzek [9]). It provides a method that shows how to group data points that populate some multidimensional space into a specific number of different clusters (cf. [102]). In Sugeno type network systems, this method partitions the data into groups, and by using statistical tools like factor analysis, it generates a fuzzy inference system with the minimum number of rules required to distinguish the fuzzy qualities associated with each of the clusters. Clustering can be compared to the k -th nearest neighborhood method which was explicitly discussed in chapter 3, section 3.1.4. The special feature of the present clustering method is that we have an additional parameter, the cluster radius, allowing to adjust the complexity of the system and thus restricting the number of rules [19]. In all our numerical examples, we indicate the ‘cluster radius’ which then essentially determines the network complexity and thus the number of network parameters and rules.

| | |
|-------------------------|--|
| Sampling Method | Uniform-/ Gaussian Latin Hypercube Samples |
| Network Type | Adaptive-Neuro Fuzzy Inference System |
| Learning Algorithm | Hybrid approach by Standard Backpropagation combined with Least Squares Method |
| Flow Solver | Fastest 2-D Finite Volume Solver Multigrid Algorithm, 6 grid levels, 40.960 control volumes on finest grid |
| Grid | Block-structured Grid |
| Optimization Strategies | - Network Feeding (NF) - Simulated Annealing (SA) - Evolutionary Network Feeding (ENF) |

Table 4.11: Methods used for numerical example

4.3.3 Numerical Results

We consider the same staggered channel as in the previous section. We divide our numerical examples by the number of design variables and in either case we varied the cluster radius, giving networks of different complexity. The network calculations are performed with the Matlab fuzzy logic toolbox [102]. An overview of all methods employed is provided in table 4.11. Table 4.12 indicates the data set used for the NF algorithm.

| Method | Design-variable | No. samples | Distribution | Interval/ Std. Dev. σ | Obj. value | in figure |
|--------|-----------------|-------------|--------------|---------------------------------|------------|-----------|
| NF | 6 | 243 | LHS Gauss | 1.5 | 5.7698 | 4.24 |
| NF | 6 | 96 | LHS Gauss | 1.5 | 5.7698 | - |
| SA | 6 | 196 | LHS Gauss | - | 5.5863 | 4.25 |

Table 4.12: Data Sets used for 6 Design Variables

We decided for 10.000 training epochs which is quite a large number and for which the corresponding ANFIS needs several (2-5) minutes. The training process stops if the designated epoch number is reached or the error goal is achieved. In all our experiments, the maximum number of training epochs has been reached. In the evolutionary strategy we then used 30 individuals in 30 generations, respectively. For all simulations which involved a network, the network output is understood as a ‘suggestion’ to the optimum. This is re-calculated with the finite volume solver giving the precise objective function value, i.e. the pressure drop.

In tables 4.13-4.17 the numerical results are summarized. In them, the number of rules correspond to the total (linear and nonlinear) fuzzy rules of each input variable. The number of parameters and rules are determined by the cluster radius which specifies the range of influence of the cluster center for each input dimension. Specifying a smaller cluster radius will usually yield more, smaller clusters in the data, and hence more rules [9, 19]. We also applied an ANFIS network without clustering - unfortunately, these networks performed too weak for our purposes and we thus do not document the corresponding results.

None of the algorithms is able to reach the DFO value of 5.5754 for the pressure drop

(see table 4.1), although the geometries found were quite similar to this global minimum. The pressure drop appears to react very sensitive on parameter variations. However, our optimization results can be considered as successful since the algorithms employed propose shape designs with corresponding pressure drop of 5.58 to 6.10 and designs that all come close the global optimum.

For the six variables case in tables 4.13-4.17, the best network result is obtained by the algorithm which employs 120 training samples, 123 checking samples and 95 parameters in total (figure (4.27)). This corresponds to a medium cluster radius and a good partition into training- and checking data samples. It is thus concluded that a careful search for network parameter is crucial for use as a successful approximation model.

For selected cases the final optimized shapes are indicated in figures (4.24)-(4.29). In them, it can be seen that figures (4.26) and (4.29) also reached reasonable shape designs. In figure (4.28), on the other hand, the channel is squeezed and corresponds to a pressure drop of 300.83. The streamlines drawn in the final shape designs indicate recirculation zones, slightly influencing the pressure drop as discussed in section 4.2.3.

The overall best optimization result is obtained by the SA algorithm with a pressure drop of 5.5863 (figure (4.25)) which comes closest to the DFO minimum. The SA data are the same as in the previous section, we state them again for clarity. Figure (4.24) corresponds to a pure scatter of input parameters, it is the best result obtained from the NF parameter sampling procedure.

Table 4.15 presents the data sets used for the more complex geometry parameterized by 12 design variables. The corresponding optimization results are summarized in tables 4.16-4.17 and figures (4.30)-(4.35). In all designs, it can be seen that energy is dissipated in recirculation zones. As the streamlines suggest in figures (4.33)-(4.34), the flow is well developed in the mid-part of the design. Figures (4.31) and (4.32) failed to give improved designs since the channel is squeezed in the upper part, causing a strong raise in the pressure drop. In this 12-parameter case, the optimization process is more complex and the results are not as close to the DFO minimum as in the previous case. However, we also see that the cluster radius is crucial for the optimization result. The ENF algorithm gives surprisingly good results compared to the NF algorithm. As in the 6-parameter example, the SA algorithm outperformed all other algorithms.

| Total | Samples | | Design- | Cluster- | Training- | Total | Objective- | in |
|-------|----------|----------|-----------|----------|-----------|------------|------------|--------|
| | Training | Checking | Variables | Radius | epochs | Parameters | value | figure |
| 243 | 200 | 43 | 6 | .7 | 10000 | 57 | 5.9032 | 4.26 |
| 243 | 120 | 123 | 6 | .7 | 10000 | 57 | 5.9419 | - |
| 243 | 200 | 43 | 6 | .5 | 10000 | 152 | 6.0036 | - |
| 243 | 120 | 123 | 6 | .58 | 10000 | 95 | 5.8764 | 4.27 |
| 243 | 220 | 23 | 6 | .45 | 10000 | 228 | 6.0181 | - |
| 243 | 220 | 23 | 6 | .4 | 30000 | 437 | 6.1602 | - |
| 243 | 120 | 123 | 6 | .45 | 10000 | 361 | 6.2570 | - |
| 243 | 120 | 123 | 6 | .4 | 30000 | 779 | 6.7693 | - |

Table 4.13: ANFIS performance for 243 training samples from Network Feeding

| Total | Samples Training | Checking | Design- Variables | Cluster- Radius | Training- epochs | Total Parameters | Objective- value | in figure |
|-------|---------------------|----------|----------------------|--------------------|---------------------|---------------------|---------------------|--------------|
| 96 | 80 | 16 | 6 | .9 | 10000 | 38 | 6.0768 | - |
| 96 | 80 | 16 | 6 | .5 | 10000 | 627 | 300.82 | 4.28 |
| 96 | 60 | 36 | 6 | .95 | 10000 | 38 | 6.0513 | 4.29 |
| 96 | 60 | 36 | 6 | .9 | 10000 | 38 | 6.0420 | - |
| 96 | 88 | 8 | 6 | .9 | 20000 | 38 | 6.0742 | - |
| 96 | 88 | 8 | 6 | .75 | 20000 | 57 | 6.5487 | - |

Table 4.14: ANFIS performance for 96 training samples from Network Feeding

| Method | Design- variables | Samples | Distribution | Interval/ Std. Dev. σ | Obj. value | in figure |
|--------|----------------------|---------|--------------|---------------------------------|------------|-----------|
| NF | 12 | 195 | LHS Uniform | $[-3, 3]$ | 5.8244 | - |
| NF | 12 | 220 | LHS Gauss | 1.0 | 5.7506 | - |
| NF | 12 | 415 | cumulated | - | - | - |
| ENF | 12 | 138 | LHS Uniform | $[-2, 2]$ | 5.7013 | - |
| ENF | 12 | 110 | LHS Gauss | 1.0 | 5.7741 | - |
| ENF | 12 | 248 | cumulated | $[-2, 2]$ | 5.7013 | - |
| SA | 12 | 215 | LHS Gauss | - | 5.6266 | 4.30 |
| SA | 12 | 118 | LHS Gauss | - | 5.6804 | - |

Table 4.15: Data Sets used for Test Problem with 12 Design Variables

| Total | Samples Training | Checking | Design- Variables | Cluster- Radius | Training- epochs | Total Parameters | Objective- value | in figure |
|-------|---------------------|----------|----------------------|--------------------|---------------------|---------------------|---------------------|--------------|
| 415 | 170 | 245 | 12 | .55 | 10000 | 74 | 6.7294 | 4.31 |
| 415 | 250 | 165 | 12 | .5 | 10000 | 74 | 27.9316 | 4.32 |
| 220 | 190 | 30 | 12 | .95 | 10000 | 74 | 7.9634 | 4.33 |
| 220 | 140 | 80 | 12 | .8 | 10000 | 111 | 5.8625 | - |
| 220 | 140 | 80 | 12 | .77 | 10000 | 296 | 9.5804 | - |
| 220 | 190 | 30 | 12 | .8 | 10000 | 148 | 6.3214 | - |

Table 4.16: ANFIS performance for training samples from NF process

| Total | Samples Training | Checking | Design- Variables | Cluster- Radius | Training- epochs | Total Parameters | Objective- value | in figure |
|-------|---------------------|----------|----------------------|--------------------|---------------------|---------------------|---------------------|--------------|
| 248 | 220 | 28 | 12 | .215 | 10000 | 74 | 6.2006 | - |
| 248 | 220 | 28 | 12 | .175 | 10000 | 111 | 6.1546 | 4.34 |
| 248 | 170 | 78 | 12 | .25 | 10000 | 74 | 6.1437 | 4.35 |
| 248 | 170 | 78 | 12 | .18 | 10000 | 89 | 174.0875 | - |

Table 4.17: ANFIS performance for training samples from ENF process